—— Confidential ——

zetera

Zetera White Paper on

Constructing µSAN™ Using HTTP

An overview, compare and contrast, technical paper on using HTTP to implement the functions of a µSAN™ instead of IP addressing.

This document references the µSAN™ White Paper (Version .35) and assumes the reader is familiar with the protocol.

Version 0.10

Author:
Charles W. (Bill) Frank
CTO, Zetera Corporation

January 2003

10-7
363372

Revision History

Version 0.1	October 2002	Original Release

Contents

Overview ____ 4

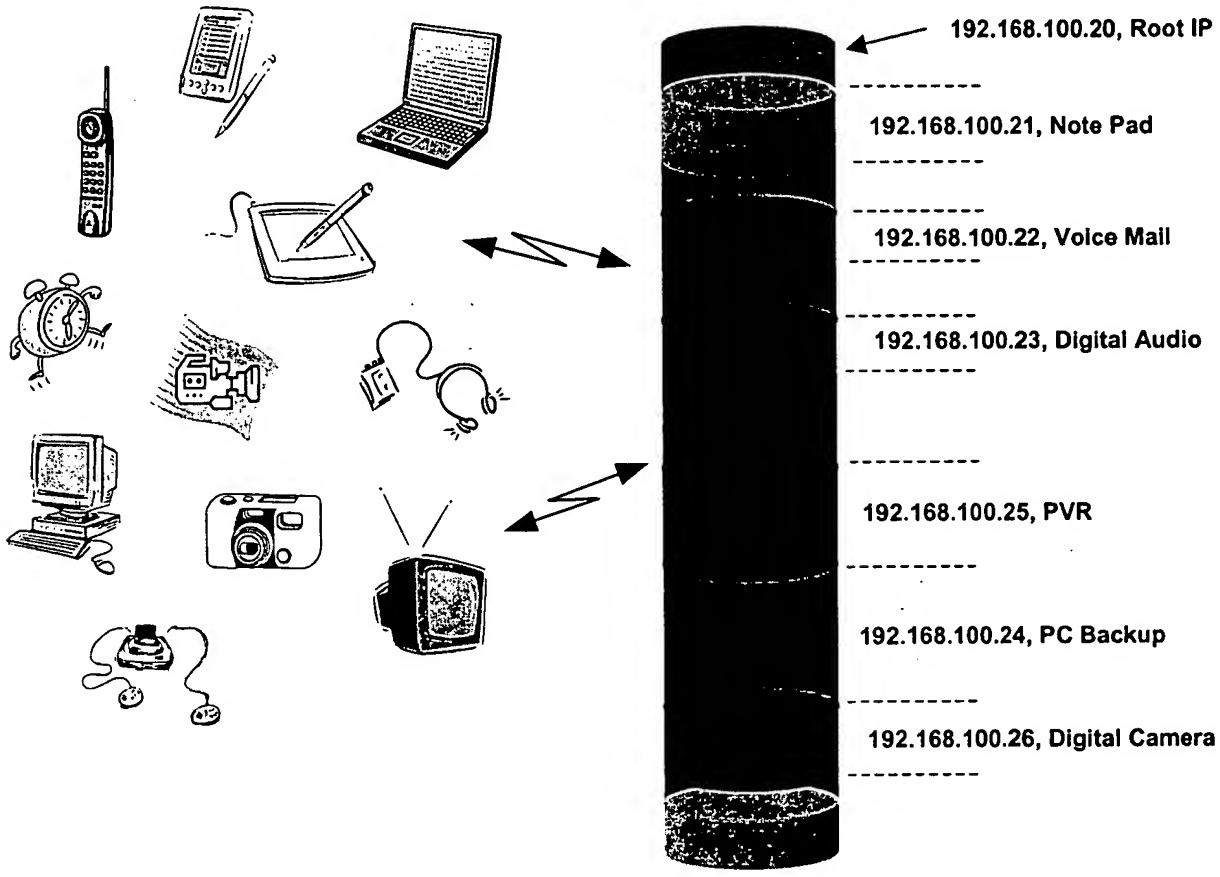
HTTP Architecture ____ 5

Considerations Regarding the use of HTTP ____ 9

Background on HTTP ____ 11

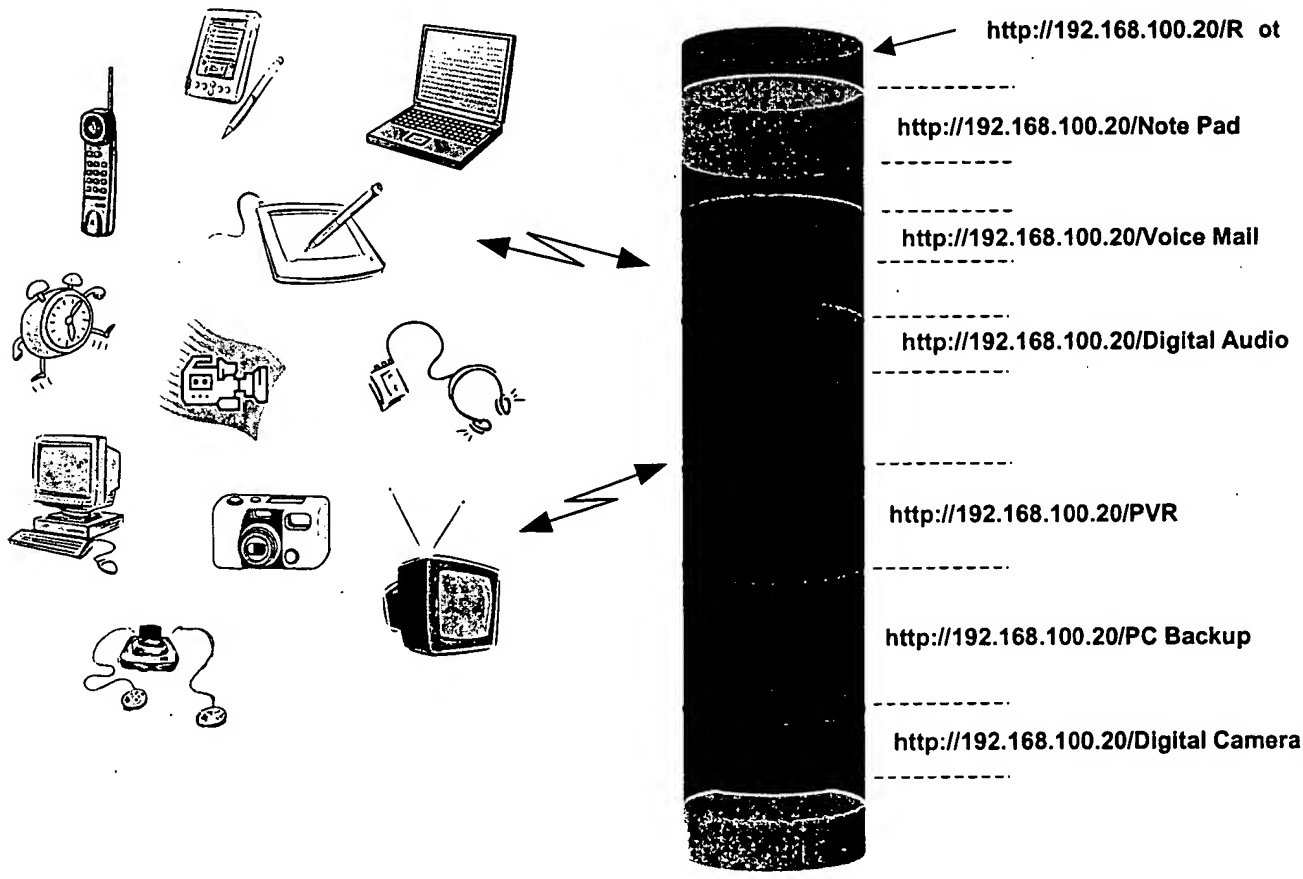
Overview

The μSAN™ White Paper identifies a storage system and protocol wherein each of the individual partitions within that storage system is addressed in the Internetworking Protocol (IP) stack by a mutually exclusive IP address as shown below. The advantages of using the IP address for partition identification lies in its exclusivity in the IP stack allowing for simple routability, masterless control and unlimited multicast associations.



HTTP Architecture

This paper identifies an alternative architecture wherein instead of using IP addresses to access individual partitions of a μSAN™ storage appliance and μSAN™ protocol for communications, the protocol uses HTTP to access virtual partitions on a block storage device comprised of a set of files addressed by a file system. The set of files is managed by an HTTP Server that establishes the virtual partition structure and manages all rules for authentication, access and sharing of the resources within the partitions. For discussion, the new network topology would look as follows:



Considerations Regarding the use of HTTP

There are several major contrasts to μSAN™:

- This is not a true SAN structure. The storage is not raw block level storage but is preformatted to support the creation and use of files.
- The operation of the system is not Peer-to-Peer but is actually Client/Server.

Advantages to HTTP:

- Clients could use HTML and XML or .NET tools to create Web page dialogs with the HTTP server. This could provide a rich set of capabilities and promote rapid development of client code.
- HTTP is routable over IP

Disadvantages to HTTP:

- The syntactic overhead associated with data transfer is very inefficient.
- HTTP is a relatively complex application running above TCP/IP requiring more resources to achieve sufficient performance.
- Access mechanisms within the HTTP server require translation to the underlying file structure and are less efficient than the IP addressed μSAN™.
- It is difficult to project HTTP into the PC architecture as a storage interface. It would be impossible to format a virtual HTTP partition in a non native format since the underlying format is required to operate,

Background on HTTP

HTTP stands for **Hypertext Transfer Protocol**. It's the network protocol used to deliver virtually all files and other data (collectively called *resources*) on the World Wide Web, whether they are HTML files, image files, query results, or anything else. Usually, HTTP takes place through TCP/IP sockets.

A browser is an *HTTP client* because it sends requests to an *HTTP server* (Web server), which then sends responses back to the client. The standard (and default) port for HTTP servers to listen on is 80, though they can use any port.

HTTP is used to transmit *resources*, not just files. A resource is some chunk of information that can be identified by a URL (it's the **R** in **URL**). The most common kind of resource is a file, but a resource may also be a dynamically-generated query result, the output of a CGI script, a document that is available in several languages, or something else. In the case of simulating a μSAN™ the HTTP server responds to requests from clients by granting or denying HTTP requests to virtual partitions.

Structure of HTTP Transactions

Like most network protocols, HTTP uses the client-server model: An *HTTP client* opens a connection and sends a *request message* to an *HTTP server*; the server then returns a *response message*, usually containing the resource that was requested. After delivering the response, the server closes the connection (making HTTP a *stateless* protocol, i.e. not maintaining any connection information between transactions).

The format of the request and response messages are similar, and English-oriented. Both kinds of messages consist of:

- an initial line,
- zero or more header lines,
- a blank line (i.e. a CRLF by itself), and
- an optional message body (e.g. a file, or query data, or query output).

Put another way, the format of an HTTP message is:

```
<initial line, different for request vs. response>
Header1: value1
Header2: value2
Header3: value3

<optional message body goes here, like file contents or query data;
 it can be many lines long, or even binary data $&*%@!^$@>
```

Initial lines and headers should end in CRLF, though you should gracefully handle lines ending in just LF. (More exactly, CR and LF here mean ASCII values 13 and 10, even though some platforms may use different characters.)

Initial Request Line

The initial line is different for the request than for the response. A request line has three parts, separated by spaces: a *method* name, the local path of the requested resource, and the version of HTTP being used. A typical request line is:

```
GET /path/to/file/index.html HTTP/1.0
```

Notes:

- The Method field is the first part of the request line and is always in uppercase.
- The path is the part of the URL after the host name, also called the *request URI* (a URI is like a URL, but more general).
- The HTTP version always takes the form "**HTTP/x.x**", uppercase.

Methods

The GET Method

GET is the most common HTTP method; it says "give me this resource".

The GET method can also be used to submit forms. The form data is URL-encoded and appended to the request URI.

To retrieve the file at the URL

```
http://www.somehost.com/path/file.html
```

First the client opens a socket to the host **www.somehost.com**, port 80 (use the default port of 80 because none is specified in the URL). Then using the GET method, send something like the following through the socket:

```
GET /path/file.html HTTP/1.0
From: someuser@jmarshall.com
User-Agent: HTTPTool/1.0
[blank line here]
```

The server should respond with something like the following, sent back through the same socket:

```
HTTP/1.0 200 OK
Date: Fri, 31 Dec 1999 23:59:59 GMT
Content-Type: text/html
Content-Length: 1354

<html>
<body>
```

```
<h1>Happy New Millennium!</h1>
(more file contents)
  .
  .
  .
</body>
</html>
```

After sending the response, the server closes the socket.

The HEAD Method

A HEAD request is just like a GET request, except it asks the server to return the response headers only, and not the actual resource (i.e. no message body). This is useful to check characteristics of a resource without actually downloading it, thus saving bandwidth. Use HEAD when you don't actually need a file's contents.

The response to a HEAD request must *never* contain a message body, just the status line and headers.

The POST Method

A POST request is used to send data to the server to be processed in some way, like by a CGI script. A POST request is different from a GET request in the following ways:

- There's a block of data sent with the request, in the message body. There are usually extra headers to describe this message body, like `Content-Type:` and `Content-Length:`.
- The *request URI* is not a resource to retrieve; it's usually a program to handle the data you're sending.
- The HTTP response is normally program output, not a static file.

The most common use of POST, by far, is to submit HTML form data to CGI scripts Here's a typical form submission, using POST:

```
POST /path/script.cgi HTTP/1.0
From: frog@jmarshall.com
User-Agent: HTTPTool/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 32

home=Cosby&favorite+flavor=flies
```

You can use a POST request to send whatever data you want, not just form submissions. Just make sure the sender and the receiving program agree on the format.

Initial Response Line (Status Line)

The initial response line, called the *status line*, also has three parts separated by spaces: the HTTP version, a *response status code* that gives the result of the request, and an English *reason phrase* describing the status code. Typical status lines are:

```
HTTP/1.0 200 OK
```

or

```
HTTP/1.0 404 Not Found
```

Notes:

- The HTTP version is in the same format as in the request line, "**HTTP/x.x**".
- The status code is meant to be computer-readable; the reason phrase is meant to be human-readable, and may vary.
- The status code is a three-digit integer, and the first digit identifies the general category of response:
    - **1xx** indicates an informational message only
    - **2xx** indicates success of some kind
    - **3xx** redirects the client to another URL
    - **4xx** indicates an error on the client's part
    - **5xx** indicates an error on the server's part

The most common status codes are:

`200 OK`
: The request succeeded, and the resulting resource (e.g. file or script output) is returned in the message body.

`404 Not Found`
: The requested resource doesn't exist.

`301 Moved Permanently`
`302 Moved Temporarily`
`303 See Other` *(HTTP 1.1 only)*
: The resource has moved to another URL (given by the **`Location:`** response header), and should be automatically retrieved by the client. This is often used by a CGI script to redirect the browser to an existing file.

`500 Server Error`
: An unexpected server error. The most common cause is a server-side script that has bad syntax, fails, or otherwise can't run correctly.

Header lines provide information about the request or response, or about the object sent in the message body.

The header lines are in the usual text header format, which is: one line per header, of the form "`Header-Name: value`", ending with CRLF. As noted above, they should end in CRLF, but should handle LF correctly.

- The header name is not case-sensitive (though the value may be).
- Any number of spaces or tabs may be between the ":" and the value.
- Header lines beginning with space or tab are actually part of the previous header line, folded into multiple lines for easy reading.

Thus, the following two headers are equivalent:

```
Header1: some-long-value-1a, some-long-value-1b
HEADER1:    some-long-value-1a,
            some-long-value-1b
```

- HTTP 1.0 defines 16 headers, though none are required. HTTP 1.1 defines 46 headers, and one (`Host:`) is required in requests. These headers help webmasters troubleshoot problems. They also reveal information about the user.

An HTTP message may have a body of data sent after the header lines. In a response, this is where the requested resource is returned to the client (the most common use of the message body), or perhaps explanatory text if there's an error. In a request, this is where user-entered data or uploaded files are sent to the server.

If an HTTP message includes a body, there are usually header lines in the message that describe the body. In particular,

- The `Content-Type:` header gives the MIME-type of the data in the body, such as `text/html` or `image/gif`.
- The `Content-Length:` header gives the number of bytes in the body.

HTTP Proxies

An *HTTP proxy* is a program that acts as an intermediary between a client and a server. It receives requests from clients, and forwards those requests to the intended servers. The responses pass back through it in the same way. Thus, a proxy has functions of both a client and a server.

Proxies are commonly used in firewalls, for LAN-wide caches, or in other situations.

When a client uses a proxy, it typically sends all requests to that proxy, instead of to the servers in the URLs. Requests to a proxy differ from normal requests in one way: in the

first line, they use the complete URL of the resource being requested, instead of just the path. For example,

```
GET http://www.somehost.com/path/file.html HTTP/1.0
```

That way, the proxy knows which server to forward the request to (though the proxy itself may use another proxy).